ASSIGNMENT 2

FINITE ELEMENT FOR FLUIDS

PRADEEP KUMAR BAL

ADVISOR: - Prof. Pablo Saez

Problem Statement:-

To solve a one-dimensional transient pure convection problem

$$u_t + au_x = 0; a = 1$$

With the given initial conditions (**Problem =2** is considered here, in which it is a product of a square wave and a sinusoidal wave) and the homogeneous Dirichlet boundary conditions which are given as u (0, t) =0 and u (L, t) =0.

Goal:-

- To solve the given problem using the Leap-frog method, the third order (TG3) and the two-step third-order Taylor-Galerkin method (TG3-2S)
- To use α=1/9 in the TG3-2S method to reproduce the phase-speed characteristics of the TG3 scheme

Solution:-

Leap Frog Method:-

$$\frac{u^{n+1} - u^{n-1}}{2\Delta t} = u_t^n = s^n - \boldsymbol{a} * \boldsymbol{\nabla} u^n$$

Weighted Residual:-

$$(w, \frac{u^{n+1}}{2\Delta t}) = (w, \frac{u^{n-1}}{2\Delta t}) + (w, s^n - \boldsymbol{a} \cdot \boldsymbol{\nabla} u^n)$$

Galerkin Formulation:-

$$(w, \frac{u^{n+1}}{2\Delta t}) = (w, \frac{u^{n-1}}{2\Delta t} + s^n) + (\boldsymbol{a} \cdot \boldsymbol{\nabla} w, u^n) - (w, u^n (\boldsymbol{a} . \boldsymbol{n}))_{\Gamma^{out}} + (w, h^n)_{\Gamma_N^{in}};$$

s=0; $h_{inlet} = -\boldsymbol{a}u.\boldsymbol{n} = \boldsymbol{0}$ as u^n at the inlet = 0; $h_{outlet} = 0$, $u^n{}_{\Gamma out} = 0$;

For the given 1D problem it is reduced to:-

$$(w, \frac{u^{n+1}}{2\Delta t}) = (w, \frac{u^{n-1}}{2\Delta t}) + (aw_x, u^n)$$

To evaluate the u^1 the first iteration is performed using the Lax-Wendroff method. At intermediate and short wavelengths the phase error of the LF scheme becomes positive as the time step is increased and deteriorates. The relative phase errors of the leap-frog schemes combined with linear finite elements using a consistent mass representation are reported in Figures 1(a) and 1(b), for the C=0.5 and C=0.9 respectively. It can be observed that the phase response of this method deteriorates as the Courant number increases. Furthermore, in the case of the leap-frog method, the combination with finite elements using a consistent mass matrix leads to a reduced stability range, namely $C^2<1/3$.

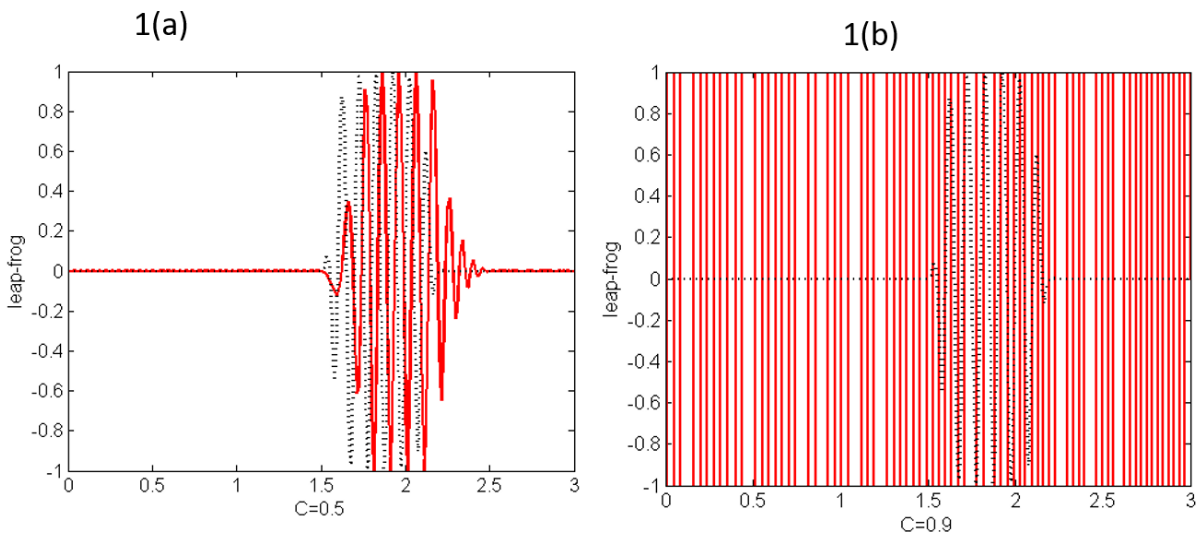

Figure 1: Solutions obtained using the Leap Frog method for the C= 0.5 and 0.9 respectively

TG3:

The linear system to be solved in this method at each time step has characteristics similar to the one obtained by applying the second-order Lax-Wendroff scheme with finite elements. The advantage here is that, with similar computational cost, we obtain third-order accuracy. The necessary condition for the numerical stability in 1D is C < 1 for TG3 as compared with $C^2 <$ 1/3 for the Lax-Wendroff and the Leapfrog finite element methods.

TG3 Scheme:-

$$\frac{u^{n+1}-u^n}{\Delta t} = u_t^n + \frac{\Delta t}{2} u_{tt}^n + \frac{\Delta t^2}{6} u_{ttt}^n + o(\Delta t^3);$$

After simplification:-

$$\left[1-\frac{\Delta t^2}{6}(\boldsymbol{a}.\boldsymbol{\nabla})^2\right]\frac{u^{n+1}-u^n}{\Delta t} = -(\boldsymbol{a}.\boldsymbol{\nabla})u^n + \frac{\Delta t}{2}(\boldsymbol{a}.\boldsymbol{\nabla})^2 u^n + s^n + \frac{\Delta t}{2}\left(s_t^n - (\boldsymbol{a}.\boldsymbol{\nabla})s^n\right) + \frac{\Delta t^2}{6}\left(s_{tt}^n - (\boldsymbol{a}.\boldsymbol{\nabla})s_t^n\right)$$

For the given problem s=0; $h_{inlet} = -\boldsymbol{a}u.\boldsymbol{n} = \mathbf{0}$ **as** u^n at inlet = 0; $h_{outlet} = 0$

$\Delta u^n{}_{\Gamma out} = 0$;

Galerkin Formulation:-

The Galerkin formulation of the given 1D problem for this scheme becomes:

$$\left(w, \frac{\Delta u}{\Delta t}\right) + \frac{\Delta t^2}{6}\left(aw_x\, , \boldsymbol{a}.\boldsymbol{\nabla}\frac{\Delta \boldsymbol{u}}{\Delta \boldsymbol{t}}\right) = \left(aw_x\, , u^n - \frac{\Delta t}{2}\, \boldsymbol{a}.\boldsymbol{\nabla}\, u^n\right)$$

The TG3 scheme possesses the unit CFL property and its accuracy characteristics are illustrated in the Figure 2. It depicts the enhanced phase accuracy. Unfortunately, the TG3 scheme experiences a drastic reduction of its stability range in multidimensional situations.

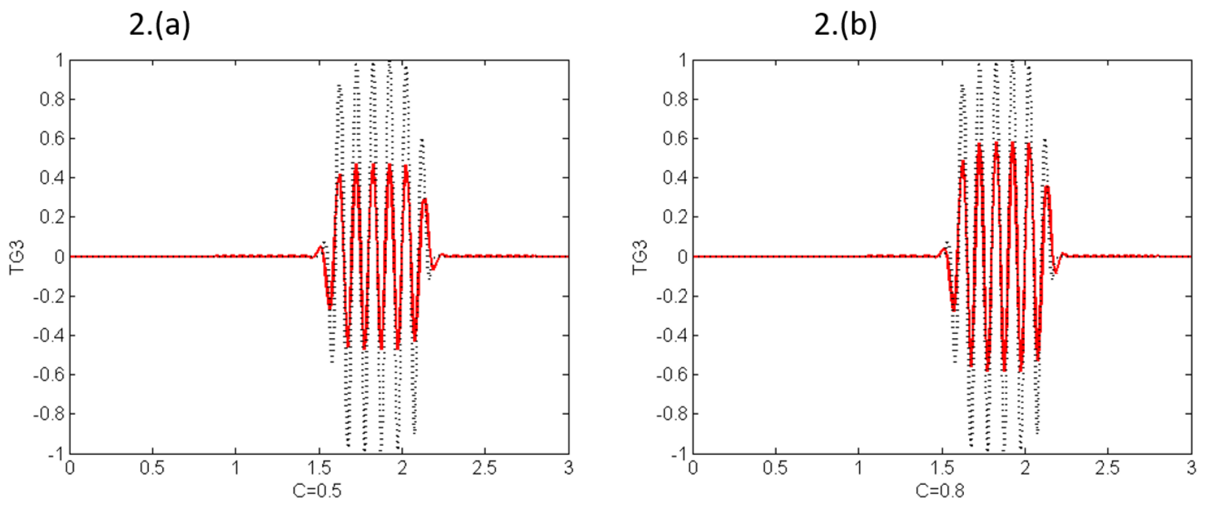

Figure 2: Solutions obtained using the TG3 method for the C= 0.5 (Fig 2. (a)) and C= 0.8 (Fig 2. (b)) respectively

TG3-2S

The two-step versions of the explicit Taylor—Galerkin method TG3 which include second time derivatives only and are thus easier to implement than the one-step method TG3, especially for solving nonlinear multidimensional hyperbolic problems. A further advantage of the two-step Taylor-Galerkin methods is their extended stability range in multidimensional situations as compared with the one-step TG3 method.

Scheme:

$$\tilde{u}^n = u^n + \frac{\Delta t}{3}\, u_t^n + \alpha\, \Delta t^2 u_{tt}^n$$

$$u^{n+1} = u^n + \Delta t\, u_t^n + \frac{\Delta t^2}{2}\, \tilde{u}_{tt}^n$$

Galerkin Formulation for the given problem:-

$$\langle w, \frac{\tilde{u}^n - u^n}{\Delta t}\rangle = \frac{1}{3}\langle \boldsymbol{a}.\nabla w, u^n\rangle - \alpha * \Delta t * \langle \boldsymbol{a}.\nabla w, \boldsymbol{a}.\nabla u^n\rangle$$

$$\langle w, \frac{u^{n+1} - u^n}{\Delta t}\rangle = \langle \boldsymbol{a}.\nabla w, u^n\rangle - \frac{\Delta t}{2}\langle \boldsymbol{a}.\nabla w, \boldsymbol{a}.\nabla \tilde{u}^n\rangle$$

Third-order accuracy is achieved when combining both steps; the parameter α only influences the coefficient of the fourth-order term in the overall time series. As a consequence, its value will only affect the modulus of the amplification factor of the resulting scheme but not its phase. For the choice α = 1/9 the two-step procedure reproduces exactly the phase-speed characteristics of the single-step TG3 scheme. It can be observed from the comparison of the Figures 2 and the Figures 3.

The condition of numerical stability for the two-step method is $|C| < \sqrt{3/4}$. The comparison of the stability limits of TG3 and TG3-2S in 1D can be observed from the Figure 4.

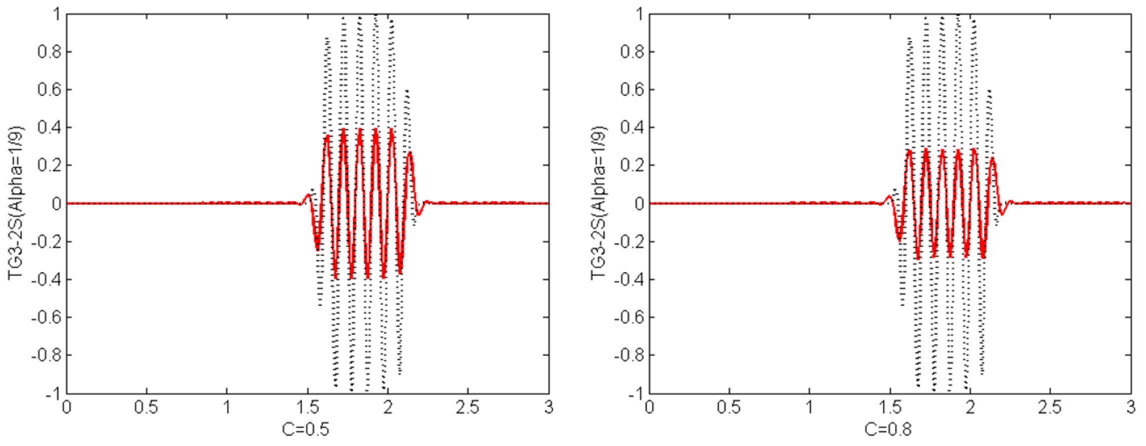

Figure 3: Solutions obtained using the TG3-2S method for the C= 0.5 and 0.8 respectively

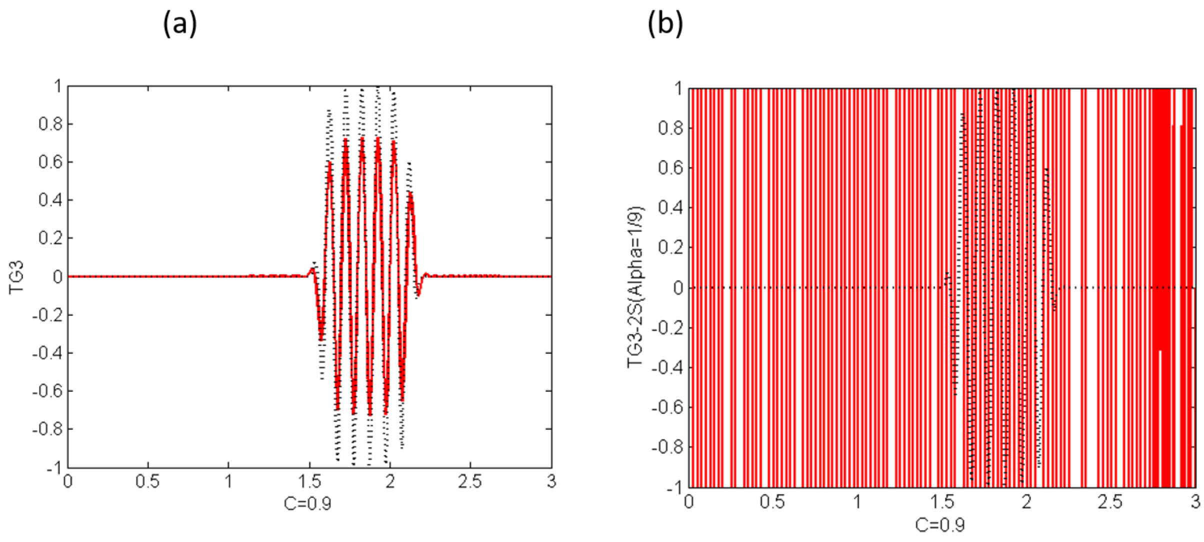

Figure 4: (a) Solutions obtained using the TG3 (C=0.9); (b) using TG3-2S (C=0.9)

The stability range of the two-step scheme in 2D remains practically unaltered with respect to that in 1D. This is in sharp contrast with the one-step Taylor-Galerkin scheme (TG3), which experiences a drastic reduction of its stability limit in multidimensional situations. Thus, the two-step formulation of the explicit Taylor-Galerkin method, besides making high-order accuracy accessible for truly nonlinear problems, offers the additional advantage of giving an isotropic stability domain in multidimensional problems. For $\alpha = 1/12$, the two-step fourth-order method is obtained. This two-step method has the same stability and accuracy properties as the classical fourth-order explicit Runge—Kutta method. This method is stable up to C= 1.

CODE:-

1. LEAP-FROG:

```
function [A,B,f] = system_LF(xnode,a)
% [A,B,f] = system_LF(xnode,a)
% L.h.s (A) and r.h.s (B,f) matrices for the leap-frog method.
% The spatial discretization is performed using linear finite
% elements and the Galerkin formulation.
%    xnode: nodal coordinates
%    a :    convection velocity
%

global dt

dtdt = 2*dt;

% Gauss points and weights on the reference element [-1,1]
xipg = [-1/sqrt(3) 1/sqrt(3)]';
wpg = [1 1]';

% Shape functions on the reference element
N_mef   =  [(1-xipg)/2 (1+xipg)/2];
Nxi_mef =  [-1/2 1/2; -1/2 1/2];

% Total number of nodes and elements
numnp = size(xnode,2);
numel = numnp-1;

% Number of Gauss points in each element
ngaus = size(wpg,1);

% Allocate storage
A = zeros(numnp,numnp);
B = zeros(numnp,numnp);
f = zeros(numnp,1);

% MATRICES COMPUTATION
% Loop on elements
for i=1:numel
    unos = ones (ngaus,1);
    h = xnode(i+1)-xnode(i);
    xm = (xnode(i)+xnode(i+1))/2;
    weight = wpg*h/2;
    isp = [i i+1];
    % Loop on Gauss points (numerical quadrature)
    for ig=1:ngaus
        N = N_mef(ig,:);
        Nx = Nxi_mef(ig,:)*2/h;
        w_ig = weight(ig);
        x = xm + h/2*xipg(ig); % x-coordinate of Gauss point
        % Matrices assembly
        A(isp,isp) = A(isp,isp) + w_ig*N'*N;
        B(isp,isp) = B(isp,isp) - w_ig*dtdt*(N')*a*Nx;
        f(isp) = f(isp) + dtdt*w_ig*(N')*SourceTerm(x);
    end
end
```

Solution Step:-

```
  if meth==4
        % Leap-frog is a two-steps method
        % first iteration is performed using Lax-Wendroff method
        if n == 1
            [A1,B1,f1] = system_LW(xnode,a);
            Atot1 = [A1 Accd';Accd zeros(2)];
            btot = [B1*u(:,n)+f1; bccd];
            aux  = Atot1\btot;
            u(:,n+1) = u(:,n) + aux(1:numnp);
            clear A1 B1 f1 Atot1
        else
            btot = [B*u(:,n)+f; bccd];
            aux  = U\(L\btot);
            u(:,n+1) = u(:,n-1) + aux(1:numnp);
        end
```

2. TG 3

```
 function [A,B,f] = system_TG3 (xnode,a)

global dt

dt_2 = dt/2;
dt2_6 = dt^2/6;

% Gauss points and weights on the reference element [-1,1]
xipg = [-1/sqrt(3) 1/sqrt(3)]';
wpg = [1 1]';

% Shape functions on the reference element
N_mef   =  [(1-xipg)/2 (1+xipg)/2];
Nxi_mef =  [-1/2 1/2; -1/2 1/2];

% Total number of nodes and elements
numnp = size(xnode,2);
numel = numnp-1;

% Number of Gauss points in each element
ngaus = size(wpg,1);

% Allocate storage
A = zeros(numnp,numnp);
B = zeros(numnp,numnp);
f = zeros(numnp,1);

% MATRICES COMPUTATION
% Loop on the elements
for i=1:numel
    unos = ones (ngaus,1);
    h = xnode(i+1)-xnode(i);
    xm = (xnode(i)+xnode(i+1))/2;
    weight = wpg*h/2;
    isp = [i i+1];
    % Loop on Gauss points (numerical quadrature)
    for ig = 1:ngaus
        N = N_mef(ig,:);
        Nx = Nxi_mef(ig,:)*2/h;
        w_ig = weight(ig);
        x = xm + h/2*xipg(ig); % x-coordinate of Gauss point
        % Matrices assembly
        A(isp,isp) = A(isp,isp) + w_ig*(N'*N+dt2_6*(a*Nx)'*(a*Nx));
        B(isp,isp) = B(isp,isp) + w_ig*dt*(a*Nx)'*(N-dt_2*a*Nx);
        f(isp) = f(isp) + w_ig*(N + dt_2*a*Nx)'*SourceTerm(x);
    end
end
```

3. TG3-2S:

```
function [A1,B1,f1,A2,B2,f2,C2] = system_TG32S (xnode,a)
% [A1,B1,f1,A2,B2,f2,C2] = system_TG32S(xnode,a) TWO STEP
% The spatial discretization is performed using linear finite
% elements and the Galerkin formulation.
%    xnode: nodal coordinates
%    a :    convection velocity
%

global dt
alpha=1/9;    %%ALPHA= 1/9 (TG3); ALPHA 1/12 ( TG4)
dt_2 = dt*dt/2;
dt2_alpha = dt^2*alpha; %%%%%%%%%%%%% ALPHA

% Gauss points and weights on the reference element [-1,1]
xipg = [-1/sqrt(3) 1/sqrt(3)]';
wpg = [1 1]';

% Shape functions on the reference element
N_mef   =  [(1-xipg)/2 (1+xipg)/2];
Nxi_mef =  [-1/2 1/2; -1/2 1/2];

% Total number of nodes and elements
numnp = size(xnode,2);
numel = numnp-1;

% Number of Gauss points in each element
ngaus = size(wpg,1);

% Allocate storage
A1 = zeros(numnp,numnp);
B1 = zeros(numnp,numnp);
f1 = zeros(numnp,1);
A2 = zeros(numnp,numnp);
B2 = zeros(numnp,numnp);
f2 = zeros(numnp,1);
C2 = zeros(numnp,numnp);
% MATRICES COMPUTATION
% Loop on the elements
for i=1:numel
    unos = ones (ngaus,1);
    h = xnode(i+1)-xnode(i);
    xm = (xnode(i)+xnode(i+1))/2;
    weight = wpg*h/2;
    isp = [i i+1];
    % Loop on Gauss points (numerical quadrature)
    for ig = 1:ngaus
        N = N_mef(ig,:);
        Nx = Nxi_mef(ig,:)*2/h;
        w_ig = weight(ig);
        x = xm + h/2*xipg(ig); % x-coordinate of Gauss point
        % Matrices assembly
       A1(isp,isp) = A1(isp,isp) + w_ig*(N'*N);
        B1(isp,isp) = B1(isp,isp) -
w_ig*((dt/3*N'*(a*Nx))+dt2_alpha*(a*Nx)'*(a*Nx));
        f1(isp) = f1(isp) + w_ig*(N')*SourceTerm(x);

        A2(isp,isp) = A2(isp,isp) + w_ig*(N'*N);
        B2(isp,isp) = B2(isp,isp) - w_ig*(dt*N'*(a*Nx));
        f2(isp) = f2(isp) + w_ig*(N')*SourceTerm(x);
        C2(isp,isp) = C2(isp,isp) - w_ig*(dt_2*(a*Nx)'*(a*Nx));
    end
end
```

STEPS TO WRITE THE ENTIRE MATRIX:-

```
% ENTIRE MATRIX
%Atot = [A Accd';Accd zeros(2)];
%[L,U] = lu(Atot);
if meth == 7     % 2-step method
A1tot = [A1 Accd';Accd zeros(2)];
[L1,U1] = lu(A1tot);
 A2tot = [A2 Accd';Accd zeros(2)];
 [L2,U2] = lu(A2tot);
else
 Atot = [A Accd';Accd zeros(2)];
 [L,U] = lu(Atot);
end
```

SOLUTION STEPS:-

```
for n = 1 : nstep
    if meth == 7     % 2-step method
        btot = [B1*u(:,n)+ f1; bccd];
        aux =  U1\(L1\btot);
        u_m =  u(:,n) + aux(1:numnp);
        btot = [B2*u(:,n) + C2*u_m + f2; bccd];
        aux  = U2\(L2\btot);
        u(:,n+1) = u(:,n) + aux(1:numnp);
else
     btot = [B*u(:,n)+f; bccd];
            aux  = U\(L\btot);
            u(:,n+1) = u(:,n-1) + aux(1:numnp);
  end
end
```

---- END----